Conservative Scheduling: Using Predicted Variance to Improve Scheduling Decisions in Dynamic Environments

Lingyun Yang[1] Jennifer M. Schopf [2] Ian Foster[1,2]

[1]*Department of Computer Science, University of Chicago, Chicago, IL 60637*
[2]*Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439*
lyang@cs.uchicago.edu [jms, foster]@mcs.anl.gov

Abstract

In heterogeneous and dynamic environments, efficient resource usage can require mappings of tasks to resources whose performance varies both in space and in time. While adaptive domain decomposition techniques have been used to address heterogeneous resource capabilities, temporal variations in those capabilities have seldom been considered. We propose a *conservative scheduling* policy that uses information about expected future variance in resource capabilities to produce more efficient data mapping decisions. Evaluation of several new one-step-ahead and low-overhead time series prediction strategies shows that a dynamic tendency prediction model with different ascending and descending behavior performs best. We extend a one-step-ahead predictor to predict *average resource* capabilities for some future time interval and *variation* of resource capabilities over some future time interval. We present a family of stochastic scheduling algorithms that exploit such predictions when making data-mapping decisions. Experimental results with a CPU-bound astrophysics application and a GridFTP implementation demonstrate that conservative scheduling can produce execution times that are both significantly faster and less variable than other techniques.

1 Introduction

In multiuser time-shared systems, performance may vary in both time and space because of competing applications. Effective use of such heterogeneous and dynamic systems requires new approaches to performance prediction and data mapping. We present here a *conservative scheduling* technique that uses predicted mean and variance resource capacity information to make data-mapping decisions.

We proceed as follows. First, we review related work (Section 2), describe the problem (Section 3), and evaluate several new one-step-ahead and low-overhead time series prediction strategies that track recent trends by giving more weight to recent data (Section 4). Next we

extend the one-step-ahead prediction strategy to obtain *average resource* capability for some future time interval and *variation* of resource capability over some future time interval (Section 5). We then (Section 6) introduce our conservative scheduling method and describe experimental results when our strategies are applied to a CPU-bound application and a parallel data transfer implementation (Section 7). Section 8 presents our conclusions about the effectiveness of our approach for multiple runs in heterogeneous environments.

2 Related Work

Many researchers [7,10,18,22-24,32] have explored the use of time-balancing or load-balancing models to reduce application execution time in heterogeneous environments. However, their work has typically assumed that resource performance is constant or slowly changing, and thus such work does not take later variance into account. For example, Dail [10] and Liu et al. [24] use the 10-second-ahead predicted CPU information provided by the Network Weather Service (NWS) [33,34] to guide scheduling decisions. While this one-step-ahead prediction at a time *point* is often a good estimate for the next 10 seconds, it is less effective in predicting the available CPU during a longer execution.

Dinda et al. use multiple-step-ahead predictions of host load [16] and their associated error covariance information to predict the running times of tasks as confidence intervals [12]. These confidence intervals, a representation of prediction error variances, can then be used for various scheduling goals [13]. In contrast, we predict the variance of resource load itself and focus on scheduling data parallel tasks to minimize overall runtime.

Yang and Casanova [39,40] present a multiround scheduling algorithm for divisible workloads. They use system performance information collected at scheduling time to decide a workload allocation scheme. If the system status changes dramatically during execution of the

application, the scheduler is switched to a greedy algorithm that assigns more work to idle computers. However, this strategy is limited to applications whose subtasks are independent of each other. The loosely synchronous application we concerned with, on the other hand, involve communications among subtasks.

Dome [5] and Mars [20] support dynamic workload balancing through migration and make the application adaptive to the dynamic environment at runtime. But the implementation of such adaptive strategies can be complex and is not feasible for all applications.

Schopf and Berman [28] defined a *stochastic scheduling policy* based on time balancing for data-parallel applications. The idea is to allocate less work to machines with higher load variance. Their algorithm uses the mean and variation of the history information but assumes that the associated stochastic data can be described by a normal distribution, an assumption they admit is not always valid [15,26].

In our approach, we define a time-balancing scheduling strategy based on the prediction of the next interval of time and a prediction of the variance. Our aim is to achieve faster and less variable application execution time.

3 Problem Statement

Efficient execution in a distributed system can require mechanisms for the *discovery* of available resources, the *selection* of a job-appropriate subset of those resources, and the *mapping* of data or tasks onto selected resources. Here, we assume that the target set of resources is fixed, and we focus on the data-mapping problem for data-parallel computation and transfer jobs. We do *not* assume that the resources have identical or even fixed capabilities or identical underlying loads. Our goal is to achieve data assignments that balance load between resources so that each resource finishes executing at roughly the same time, thereby minimizing execution time.

This form of load balancing, also known as *time balancing,* is generally accomplished by solving a set of equations, such as the following, to determine the data assignments:

$$E_i(D_i) = E_j(D_j) \quad \forall i, j \tag{1}$$

$$\sum D_i = D_{Total}.$$

D_i is the amount of data assigned to resource i; D_{Total} is the total amount of data for the job; and $E_i(D_i)$ is the execution time of task on resource i and is generally parameterized by the amount of data on D_i. It can be calculated by using a performance model of the task such as the following (note that the performance of an application can be affected by the future capacity of both the network bandwidth behavior and the CPU availability):

$$E_i(D_i) = Comm(D_i)*(futureNWCapacity) + Comp(D_i)*(futureCPUCapacity).$$

To proceed, we need mechanisms for (a) obtaining some measure of future capability and (b) translating this measure into an effective resource capability. Two measures of future resource capability are important: the expected value and the expected variance in that value. One approach to obtaining these two measures would be to negotiate a service level agreement (SLA) with the resource owner to contract to provide the specified capability [9]. Or, we could use historical data to predict future behavior [12,27,29,31,33,34,36]. We focus in this article on the latter approach but emphasize that our results for topic (b) are also applicable in the SLA case.

4 One-Step-Ahead Prediction Strategies Study

We present two families of strategies: (1) homeostatic prediction and (2) tendency-based prediction. Each strategy predicts the one-step-ahead value based on a fixed number of immediately preceding history data measured at a constant-width time interval. We use the following notation: V_T is the measured value at the Tth measurement; P_{T+1} is the predicted value for measurement value V_{T+1}; and N is the number of history data points used in the prediction.

4.1 Homeostatic Prediction Strategies

"Homeostatic prediction strategies" work on the assumption that if the current value is greater (less) than the mean of history values, then the next value is likely to decrease (increase). More formally, this kind of strategy can be expressed as follows:

if (V_T > $Mean_T$) then
    P_{T+1} = V_T – *DecrementValue;*
    [Optional DecrementValue adaptation process]
else if (V_T < $Mean_T$) then
    P_{T+1} = V_T + *IncrementValue;*
    [Optional IncrementValue adaptation process]
else
    P_{T+1} = V_T;

where $Mean_T$ is the mean of the N history data points, calculated by the following formula:

$$Mean_T = (\Sigma_{i=1..N} V_i)/N. \quad (2)$$

At every prediction step, the increment or decrement value can be an independent value or a relative value proportional to the current measurement. The increment or decrement value can be "static," such that it is fixed for all prediction steps, or "dynamic," such that it is adapted to the time series at each step. Different combinations result in four homeostatic prediction strategies: independent static, independent dynamic, relative static, and relative dynamic. We present a detailed description of these strategies next. Our selection for the parameter values for each strategy is discussed in Section 4.3.1.

4.1.1 Independent Static Homeostatic Prediction Strategy

The *independent static* homeostatic strategy generates a prediction by changing the current value by a fixed amount, without any adaptation process. The decrement (increment) constant may change depending on the training set. Values between 0.05 and 1 are reasonable.

4.1.2 Independent Dynamic Homeostatic Prediction Strategy

The *independent dynamic* homeostatic strategy dynamically adjusts the amount of the increment or decrement value by means of an adaptation process:

```
Measure V_{T+1};                    // DecrementValue adaptation process:
  RealDecValue_T = V_T - V_{T+1};
  DecConstant_{T+1} = DecConstant_T + (RealDecValue_T – DecConstant_T) * AdaptDegree;
  Measure V_{T+1};                  // IncrementValue adaptation process:
  RealIncValue_T = V_{T+1} - V_T;
  IncConstant_{T+1} =IncConstant_T + (RealIncValue_T - IncConstant_T) * AdaptDegree;
```

At each time step, after we measure the real data (V_{T+1}), we calculate the difference between the current measured value and the last measured value, thus determining the real decrement (increment) we should have used in the last prediction in order to get the actual value. We adapt the value of the decrement (increment) value accordingly and use the adapted IncConstant (or DecConstant) to predict the next data point. The parameter AdaptDegree can range from 0 to 1 and expresses the adaptation degree of the variation. If AdaptDegree is equal to 0, the $DecConstant_{T+1}$ ($IncConstant_{T+1}$) is not adapted at all, and we have *nonadaptation* (or a static approach). If AdaptDegree is equal to 1, the $DecConstant_{T+1}$ ($IncConstant_{T+1}$) is equal to $RealDecValue_T$ ($RealIncValue_T$), or *full adaptation*. The goal is to obtain a value of AdaptDegree that results in minimal average error rate; values between 0.05 and 1 are reasonable.

4.1.3 Relative Static Homeostatic Prediction Strategy

The *relative static* homeostatic strategy assumes that a larger load value has more potential to change than does a smaller load value. Thus, this strategy modifies the independent static homeostatic prediction strategy so that the increment or decrement applied to a prediction is proportional to the current value instead of a constant value. The decrement (increment) value can be expressed by DecrementValue= V_T* DecrementFactor (IncrementValue= V_T* IncrementFactor). Increment or decrement values between 0.05 and 1 are reasonable.

4.1.4 Relative Dynamic Homeostatic Prediction Strategy

The *relative dynamic* homeostatic strategy alters the prediction value by a relative amount, as does the relative static homeostatic strategy, but allows the value of IncrementFactor and

DecrementFactor to be adapted dynamically, using the same method as in the independent dynamic homeostatic prediction strategy.

4.2 Tendency-based Prediction Strategies

Our second family of prediction strategies predicts the next value according to the tendency of the time series change. This approach assumes that if the current value increases, the next value will also increase and that if the current value decreases, the next value will also decrease. Formally, tendency-based prediction strategies can be expressed as follows:

```
if ((V_T - V_T-1)<0)                    //Determine Tendency
      Tendency="Decrease";
else if ((V_T-1 - V_T )<0)
     Tendency="Increase";
if (Tendency="Decrease") then
      P_T+1 = V_T – DecrementValue;
      DecrementValue adaptation process
else if (Tendency="Increase") then
      P_T+1 = V_T + IncrementValue;
      IncrementValue adaptation process
```

The variation (DecrementValue and IncrementValue) can be an independent or relative value proportional to the current value. Since the static prediction strategies always give worse results than does a simple last-value prediction strategy in the initial experiments, we exclude the static case from this discussion.

Tendency-based strategies have an additional possible source of error. Since it is impossible to predict when a time series is going to "change direction," a large error can occur at the *turning point*. To minimize this kind of error, we use the mean of the history data as the threshold value. In the increase phase, if the current data is smaller than the threshold value, the variation will be adapted normally; if the time series increases to a value that is bigger than the threshold value, the next step may be a turning point. We calculate the percentage of the history data that is greater than the current data and use this value as the possibility of current data *not* being a turning point. The larger the current value is, the more possible that it is the turning point, and

the less the percentage of the history data bigger than it is. So the IncrementValue adaptation process can be expressed in the following way.

$Mean_T = (\Sigma_{i=1..N} V_i)/N;$
$RealIncValue_T = V_{T+1} - V_T;$
$NormalInc = IncValue_T+(RealIncValue_T- IncValue_T)* AdaptDegree;$
if ($V_{T+1} < Mean_T$) // normal adaptation
    $IncrementValue_{T+1} = NormalInc;$
else
    $PastGreater_T$=(the number of past data points greater than V_T) /N;
    $TurningPointInc = IncValue_T*PastGreater_T;$
    $IncrementValue_{T+1}$=Min(abs(NormalInc), abs(TurningPointInc));

NormalInc is the value of the $IncrementValue_{T+1}$ in the case of normal adaptation. When the current value is higher than $Mean_T$, it may be a turning point, and the value of $PastGreater_T$ (the percentage of the past time series values greater than the current value) will be small (<0.5). Hence, the possibility that the current value is *not* the turning point is small, so we adjust the increment value accordingly. If we predict the value to go in the wrong direction, the error is still small. (The DecrementValue can be adapted in the same way by using the percentage of the history data smaller than current value when the current value decreases to a value smaller than the threshold value.)

4.2.1 Independent Dynamic Tendency Prediction Strategy

The independent dynamic tendency strategy predicts the next step value by adding or subtracting an independent increment or decrement value from the current value according to the tendency of the value change. For this strategy, we determined the increment and decrement values just as we did in Sections 4.1.1 and 4.1.2.

4.2.2 Relative Dynamic Tendency Prediction Strategy

The relative dynamic tendency strategy is similar to the independent dynamic tendency prediction strategy except that the increment value or decrement value is in proportion with the current value. The increment and decrement values are determined in the same way as for the relative static and dynamic homeostatic approaches, discussed in Sections 4.1.3 and 4.1.4.

4.2.3 Dynamic Tendency Prediction Strategy

In initial experiments using CPU load time series, the independent tendency prediction strategy resulted in better predictions during an increase phase and the relative tendency prediction strategy generally resulted in better predictions during a decrease phase. One possible explanation is that while a CPU time series is increasing, the independent tendency strategy better tracks the behavior because of very small increases independent of the actual value of the prediction, but that during the decrease phase the relative prediction strategy applies a value is proportional to the current value more in keeping with the trend of the load behavior. Further experiments (Section 4.3) support this tentative explanation..

Because of this initial result, we define a mixed tendency-based prediction strategy that predicts the next value for an increase phase using the independent tendency prediction strategy and for a decrease phase uses the relative tendency prediction strategy.

$DecrementValue = V_T * DecrementFactor$
$IncrementValue = IncrementConstant$

For completeness, we examined the use of the independent constant in the decrement phase and a relative value in the increment phase, but worse predictions resulted in all cases.

4.3 Prediction Strategy Evaluation

We ran two sets of experiments using our predictors. In the first set, we ran all of our predictors on a small set of time series over which we had complete control, and we evaluated the effect of different collection rates on our own predictors, on a simple last-value predictor, and on the Network Weather Service (NWS) [33,34]. In the second set, we ran a larger set of 38 load traces and evaluated only our best predictor and NWS.

The last-value predictor uses the current measured value as the predicted value of the next measurement. Harchol-Balter and Downey [21] show that this is a useful prediction strategy for

CPU resources. It has low computation and storage overhead and is the default predictor in several current systems because of its simplicity.

NWS dynamically selects the best predictor from a set that includes mean-based prediction strategies, median-based prediction strategies, and AR model-based prediction strategies. Its forecasts are equivalent to, or slightly better than, the best forecaster in the set. Hence, if our prediction strategy performs better than the NWS predictor, it can perform better than all the prediction techniques in the set.

We did no model fitting for any of the experiments, as is commonly needed in linear regression techniques. Instead, the parameters were defined by using training data off-line before the experiments, as described in Section 4.3.1. Thus, we minimized the run-time cost (on average, this is only a few milliseconds per prediction).

4.3.1 Input Parameters

To determine the input parameters, we ran 25 experiments each involving a one-hour CPU load time series, and we evaluated increment and decrement values at intervals of 0.05 between 0 and 1 using the following error formula.

$$\text{Average Error Rate} = \frac{\sum_{i=1..N} abs(P_i - V_i)/V_i}{N} * 100\% \quad (3)$$

The value that results in minimal average error rate is considered best. For our experiments, we found the best results with IncrementConstant= DecrementConstant = 0.1, IncrementFactor = DecrementFactor = 0.05, and AdaptDegree = 0.5, and used them for all of our predictions.

We also studied the sensitivity of the mixed variation prediction strategy to a selection of AdaptDegree parameter values; the details can be found at [36]. We concluded that the value of the parameter does not significantly affect the prediction capability of our strategy as long as extremes are avoided, and we therefore selected an intermediate value of 0.5 for our studies.

4.3.2 Prediction Strategy Evaluation

We ran a set of experiments on four machines to evaluate the prediction strategies presented in Section 4.1 and 4.2. For each machine, we collected one set of data (spanning roughly 28 hours) and then examined it as three different time series: 0.1 Hz (measure the data every 10 seconds) with roughly 10,000 data points; 0.05 Hz (measure the data every 20 seconds), and 0.025Hz (measure the data every 40 seconds). Detailed discussion of the properties of the time series can be found in [36].

We evaluated our time series prediction strategies on twelve CPU load time series. The error rates and the standard deviations of the prediction strategies when tested against these time series are shown in Table 1, with the best predictors shown in boldface.

All the prediction strategies gave less accurate prediction on average for the traces with lower frequency. We attribute this result to (a) data points being more widely spaced in time, so the last data points are not as "current" as the traces where there is more data, and (b) the prediction point being farther in the future. We also see that the independent static homeostatic strategy, without any dynamic adjustment, always gives the worst results.

Tendency prediction strategies outperform other prediction strategies almost in all cases. In particular, the strategy using mixed variation gives better performance on average than the other two tendency prediction strategies for time series collected from different machines. It also achieves the smallest or near-smallest standard deviation of prediction error on 12 time series. Moreover, tendency prediction with the mixed variation method outperforms the NWS predictor on all time series, with an average prediction error 20.68% less than that of the NWS predictor.

Table 1: The error of different prediction strategies, with the best in each case shown in boldface.

(1) Mean and standard deviation of the prediction errors on time series collected from abyss.cs.uchicago.edu

	0.1 Hz		0.05 Hz		0.025 Hz	
	Mean	SD	Mean	SD	Mean	SD
Independent Static Homeostatic	496.10%	4.2855	492.26%	4.3583	488.90%	4.4384
Independent Dynamic Homeostatic	12.50%	0.2369	25.51%	0.4153	56.70%	0.9756
Relative Static Homeostatic	13.40%	0.2115	24.85%	0.2771	44.37%	0.3960
Relative Dynamic Homeostatic	13.53%	0.2585	28.67%	0.6984	59.57%	1.5305
Independent Dynamic Tendency	11.42%	0.2097	21.45%	0.2742	40.44%	0.3949
Relative Dynamic Tendency	11.54%	0.2338	20.40%	0.3203	36.15%	0.4799
Mixed Tendency	**11.13%**	0.2094	**19.48%**	**0.2741**	**34.23%**	**0.3941**
Last Value	14.40%	**0.2068**	25.84%	0.2742	45.62%	0.3984
Network Weather Service	13.43%	0.2071	25.08%	0.2760	45.89%	0.4315

(2) Mean and standard deviation of the prediction errors on time series collected from vatos.cs.uchicago.edu

	0.1 Hz		0.05 Hz		0.025 Hz	
	Mean	SD	Mean	SD	Mean	SD
Independent Static Homeostatic	333.75%	4.0129	340.31%	4.0151	360.14%	3.9996
Independent Dynamic Homeostatic	12.76%	0.2067	26.19%	0.3531	66.62%	1.0480
Relative Static Homeostatic	16.46%	0.1929	30.16%	0.2561	57.52%	0.3906
Relative Dynamic Homeostatic	15.48%	0.4531	33.73%	0.8334	102.55%	3.5787
Independent Dynamic Tendency	12.38%	0.1926	22.78%	0.2583	43.16%	0.3699
Relative Dynamic Tendency	11.77%	0.2722	20.25%	0.3735	36.85%	0.5569
Mixed Tendency	**10.78%**	0.1947	**18.74%**	0.2607	**34.31%**	**0.3628**
Last Value	16.50%	**0.1879**	29.40%	**0.2510**	57.14%	0.3874
Network Weather Service	15.53%	0.1883	25.00%	0.2515	57.33%	0.3913

(3) Mean and standard deviation of the prediction errors on time series collected from mystere.ucsd.edu

	0.1 Hz		0.05 Hz		0.025 Hz	
	Mean	SD	Mean	SD	Mean	SD
Independent Static Homeostatic	158.09%	1.9350	167.71%	1.9891	185.06%	2.1680
Independent Dynamic Homeostatic	21.24%	0.2655	38.47%	0.3867	70.20%	0.5989
Relative Static Homeostatic	22.21%	**0.1929**	37.94%	0.2329	63.09%	0.3731
Relative Dynamic Homeostatic	43.81%	1.5344	85.09%	2.2558	156.26%	4.3681
Independent Dynamic Tendency	18.38%	0.2097	34.96%	0.2632	62.10%	0.4109
Relative Dynamic Tendency	29.01%	0.8312	55.81%	1.2062	103.45%	2.0504
Mixed Tendency	**17.31%**	0.2639	**32.21%**	0.3773	**55.81%**	0.5749
Last Value	19.86%	0.2045	35.56%	**0.2270**	99.47%	**0.3445**
Network Weather Service	18.88%	0.1945	34.92%	0.2288	96.96%	1.4816

(4) Mean and standard deviation of the prediction errors on time series collected from pitcairn.mcs.anl.gov

	0.1 Hz		0.05 Hz		0.025 Hz	
	Mean	SD	Mean	SD	Mean	SD
Independent Static Homeostatic	6.94%	0.0352	6.29%	0.0425	7.83%	0.0482
Independent Dynamic Homeostatic	2.54%	0.0262	4.23%	0.0407	7.70%	0.0568
Relative Static Homeostatic	2.73%	0.0248	4.45%	**0.0364**	7.17%	0.0462
Relative Dynamic Homeostatic	2.68%	0.0242	4.48%	0.0371	7.29%	0.0515
Independent Dynamic Tendency	2.43%	0.0239	4.11%	0.0365	**7.07%**	0.0476
Relative Dynamic Tendency	**2.29%**	**0.0237**	**3.91%**	0.0409	7.39%	0.0575
Mixed Tendency	**2.29%**	**0.0237**	**3.91%**	0.0409	7.38%	0.0574
Last Value	2.69%	0.0242	4.46%	**0.0364**	7.24%	**0.0473**
Network Weather Service	2.69%	0.0242	4.49%	0.0365	7.47%	0.0479

4.3.3 Varied Time-Series Comparison

We also compared the techniques on a larger set of CPU load time series collected by Dinda [14]. These week-long, 1 Hz resolution time series represent 38 different machines, including production and research cluster machines, computer servers, and desktop workstations. The time series have complex, rough, and often multimodal distributions that are not well fitted by analytic distributions such as the normal or exponential distributions. All of the time series exhibit a high degree of self-similarity and epochal behavior. Detailed statistical properties of these CPU load time series can be found in [14].

For our experiments, we selected 38 one-day time series collected on August 18, 1997. The experimental results show that the mixed tendency prediction strategy outperforms the NWS predictors on all 38 time series with different properties. Specifically, it achieves a prediction error that is 36% lower on average than that achieved by NWS.

Our experiments also showed that this predictor does not perform well on network data. Instead, the NWS predictor is the best overall. One possible explanation is that for most of the network capability time series, the autocorrelation function value between two adjacent observations is small. Our new homeostatic and tendency-based prediction strategies, which give more weight to recent data, cannot track the trend in network capability time series well. NWS predictors, taking account of more statistic information, will give better a prediction for these time series. To predict future network capability information, we therefore used NWS predictors.

5 Mean and Variance Prediction

We now describe how the time series predictor can be extended to obtain three types of predicted resource performance information: the next-step predicted resource capability at a future time *point* (Section 5.1); the *average interval* resource capability for some future time

interval (Section 5.2); and the *variation* of resource capability over some future time interval (Section 5.3).

5.1 One-Step-Ahead Resource Capability Prediction

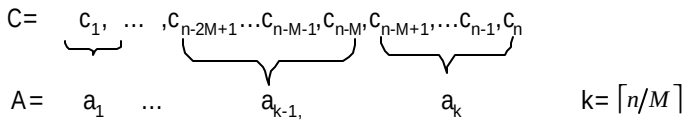
For the one-step-ahead prediction, we treat the measured preceding resource capability time series as the input to the predictor. The predictor's output is the predicted resource capability at the next step. For CPU load prediction, we use our mixed tendency strategy as the one-step-ahead predictor. For network capability prediction, we use the NWS predictor because it performs better.

5.2 Interval Resource Capability Prediction

The second type of prediction technique involves predicting the resource capability over the time interval during which an application will run. Since both the CPU load and network bandwidth time series exhibit a high degree of self-similarity [8,15], averaging values over successively larger time scales will not produce time series that are dramatically smoother. Thus, to calculate the predicted average resource capability an application will encounter during its execution, we need to first *aggregate* the original capability time series into an interval capability time series and then run predictors on this new interval time series to estimate its future value.

Aggregation consists of converting the original capability time series into an interval capability time series by combining successive data over a nonoverlapping larger time scale. The aggregation degree M is the number of original data points used to calculate the average value over the time interval. This value, which can be approximate, is determined by the resolution of the original time series and the execution time of the applications. For example, the resolution of the original time series is 0.1 Hz, or measured every 10 seconds. If the estimated application execution time is about 100 seconds, the aggregation degree is 10. In other words, 10 data points

from the original time series are needed to calculate one aggregated value over 100 seconds. The process of aggregation is

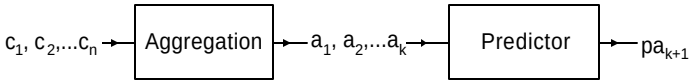


$C=c_1,c_2,\ldots,c_n$ is the original preceding capability time series measured at constant-width time interval and $A=a_1,a_2,\ a_k$ ($k=\lceil n/M \rceil$) is the interval capability time series, calculated by

$$a_i = \frac{\sum_{j=1..M} c_{n-(k-i+1)*M+j}}{M} \qquad i=1..k \ . \tag{4}$$

Each value in the interval capability time series a_i is the average resource capability over the time interval that is approximately equal to the application execution time.

Next, we use the one-step-ahead predictor on the aggregated time series to predict the mean interval capability.



The output pa_{K+1} is the predicted value of a_{k+1}, which is approximately equal to the average resource capability the application will encounter during execution.

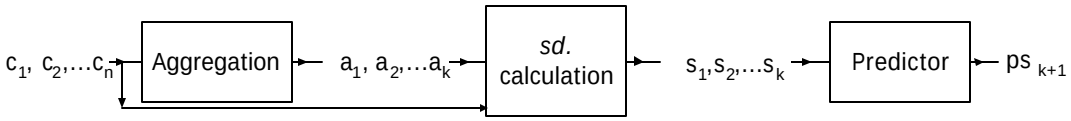
5.3 Resource Capability Variance Prediction

To predict the variation of resource capability, during the execution of an application, we calculate the standard deviation time series using the original resource capability time series *C* and the interval resource capability time series *A* (defined in Section 5.2):

$$s_i = \sqrt{\frac{\sum_{j=1..M} (c_{n-(k-i+1)*M+j} - a_i)^2}{M}} \qquad i=1..k \ . \tag{5}$$

Each value in standard deviation time series s_i is the average difference between the resource capability and the mean resource capability over the interval.

To predict the standard deviation of the resource capability, we use the one-step-ahead predictor on the standard deviation time series. The output ps_{k+1} will be the predicted value of s_{k+1}, or the predicted capability variation for the next time interval.



6 Conservative Scheduling

In this section, we show how we used the three types of predicted information from Section 5 to guide the data mapping decisions in two different contexts: a CPU-bound astrophysics application (Cactus) and a GridFTP implementation in the Globus Toolkit®.

6.1 Data Parallel Application Scheduling

We first applied our scheduling algorithms in the context of Cactus [3,4], a numerical modeling system used here to simulate a 3D scalar field produced by two orbiting astrophysical sources. This application decomposes the 3D scalar field over processors and places an overlap region on each processor. For each time step, each processor updates its local grid point and then synchronizes the boundary values. It is an iterative, loosely synchronous application. We used a one-dimensional decomposition to partition the workload. The full performance model for Cactus is described elsewhere [24], but in summary it is

$$E_i(D_i) = \text{start\_up time} + (D_i * Comp_i(0) + Comm_i(0)) * \text{slowdown(effective CPU load)}.$$

The startup time incurred when initiating computation on multiple processors in a workstation cluster was experimentally measured. $Comp_i\ (0)$ and $Comm_i\ (0)$, the computation time of per data point and communication time of Cactus in the absence of contention, was calculated by using

formulas described in [25]. The function *slowdown(effective CPU load)*, which represents the contention effect on the execution time, was calculated by using the formula described in [24].

Cactus performance is greatly influenced by the actual CPU performance achieved in the presence of contention from other competing applications. The communication time is less significant when running on a local area network, but for wide-area network experiments this factor would also be parameterized by a capacity measure. To capture the impact of contention, we defined the *effective CPU load* to be the interval load prediction (Section 5.2) plus the predicted variance (Section 5.3) in that quantity. We then used this quantity when allocating work to computers, with the effect that less work is allocated to highly varying machines.

6.2 Parallel Data Transfer Scheduling

The increasingly common practice of using multiple distributed storage systems as a distributed data store within which large datasets may be replicated has led to the problem of how to access replicated data efficiently. Multiple-source parallel transfers can improve data throughput time by fetching data from several replicas in parallel. However, we then face the problem of deciding how to distribute the data load among different storage resources.

6.2.1 Network Capability Prediction

We assume that the target set of sources is fixed, and we focus on the data allocation problem for multiple-link parallel data transfers. Our goal is to balance load between network links so that each link finishes transferring at roughly the same time. To this end, we use a time-balancing mechanism to made data assignment decisions, as accomplished by Formula 1 (Section 3).

For parallel data transfer problems, $E_i(D_i)$ is the time needed to transfer D_i data from *i*th data source to the destination. It can be calculated by using the formula $E_i(D_i) = \text{EffectiveLatency}_i + D_i/\text{EffectiveBandwidth}_i$. Thus, we first determine the value of effective network capability the

data transfer will experience during the entire transfer period. We then use this effective capability in the time-balancing formula to decide the data-mapping strategy.

Because network capability can have a large variation – sometimes twice the mean – a *tuning factor* also is needed to limit the influence of the standard deviation on the mean. To allow for the use of variation information, we define the effective bandwidth of a link as Effective Bandwidth = BandwidthMean + TF * BandwidthSD, where BandwithMean is the predicted mean bandwidth of the network link the data will encounter during transfer, BandwidthSD is the predicted variation of bandwidth of the network link the data will encounter during transfer, and TF is a per link *tuning factor* that determines how conservative the data allocation policy should be. For links with higher variation, we prefer a more conservative scheduling policy.

Note that we focus here on the bandwidth because, in our experiments, the latency is only a very small portion of the total data transfer time: < 0.1% for network links within one domain, and <1% for network links across domains.

6.2.2 The Tuning Factor

We calculate EffectiveBandwidth using a formula based on the base predicted mean bandwidth value, the tuning factor, and the standard deviation. Specifically, we vary the number of standard deviations added to the base bandwidth mean value using TF. The basic idea is to assign less data on network links with a larger variability in performance. Thus, we require a TF value that is inversely proportional to the variance of the network bandwidth. The TF value, in addition, must be able to limit the value added to the mean. We thus define TF with the algorithm in Figure 1.

```
N=SD/Mean
If (N>1)
   TF=1/(2*N^2);
Else
   TF=1/N-N/2;
```

Figure 1: The algorithm to compute our tuning factor.

This algorithm will give a TF that has the following characteristics:

- TF = 0 to ½ when SD/Mean > 1. The higher variation the network link has in its capability, the higher the N value it will have. When the standard deviation is larger than the mean of the bandwidth (SD/Mean>1), the network is considered to be high variable and less reliable. We want a smaller TF and thus a smaller effective bandwidth value.
- TF = ½ to 8 when SD/Mean <= 1. When the standard deviation is smaller than the mean of the bandwidth (N <= 1), the network link is considered to be low variable and more reliable. We want a larger TF value and thus a larger effective bandwidth value.
- In both cases, the values of TF and TF*SD are inversely proportional to N.

To illustrate our idea, we calculate the value of TF and TF*SD by our algorithm, while fixing the mean bandwidth value equal to 5 Mb/s and changing the standard deviation of bandwidth from 1 to 15. We find that the values of both TF and TF*SD are inversely proportional to the bandwidth standard deviation (and N), for a fixed mean. For network links with higher variation, we will have a smaller TF and smaller effective bandwidth value and thus a more conservative data-scheduling decision. The value added to the mean is less than the mean of the bandwidth. The validity of the tuning factor and the tuned conservative scheduling method is evaluated in the next section. However, we acknowledge that other approaches for calculating the TF value may further improve the efficiency of the tuned conservative scheduling method.

7 Conservative Scheduling Experiments

To validate our scheduling strategy, we applied it in two contexts: a CPU-bound astrophysics application and a GridFTP implementation. We conducted experiments on the GrADs [6] test bed, which comprises workstation clusters at the University of Chicago, University of Illinois at

Urbana-Champaign, University of Tennessee, University of California at San Diego, University of Houston, and University of South California's Information Sciences Institute.

7.1 Data-Parallel Application Experiments

7.1.1 Experimental Methodology

To show the efficiency of our conservative scheduling policy, we compared the execution times of the Cactus application with five scheduling policies:

(1) One-Step Scheduling (OSS): Use the one-step-ahead prediction of the CPU load, as described in Section 5.1, for the effective CPU load.

(2) Predicted Mean Interval Scheduling (PMIS): Use the interval load prediction, described in Section 5.2, for the effective CPU load.

(3) Conservative Scheduling (CS): Use the conservative load prediction.

(4) History Mean Scheduling (HMS): Use the mean of the history CPU load for the 5 minutes preceding the application start time for the value for effective CPU load. This approximates the estimates used in several common scheduling approaches [30,32].

(5) History Conservative Scheduling (HCS): Use the conservative estimate CPU load defined by adding the mean and variance of the history CPU load collected for 5 minutes preceding the application run as the effective CPU load. This approximates the prediction and algorithms used in [28].

At UIUC, we used a cluster of four 450 MHz Linux machines. At UCSD, we used a cluster of six Linux machines: four machines with a 1733 MHz CPU, one with a 700 MHz CPU, and one with a 705 MHz CPU. At Argonne, we used a cluster of thirty-two 500 MHz CPU Linux machines. All machines were dedicated during experiments.

To evaluate the scheduling policies under identical workloads, we used a load trace playback tool [17] to generate a background workload from a trace of the CPU load that results in realistic

and repeatable CPU contention behavior. We chose 64 load time series from [1] with different mean and variation. We did experiments with 10 different configurations. Complete results and discussion can be found at [37]. Representative results are analyzed in the following section.

7.1.2 Experimental Results

To compare the various policies, we used three metrics: an absolute comparison of run times, a relative measure of achievement, and a statistical analysis to show the significance of the improvement of our strategy. The first metric gives a rough valuation on the performance of each scheduling policy over a given interval of time. Over the entire run, the Conservative Scheduling policy exhibited 2%–7% less overall execution time than did the History Mean and History Conservative Scheduling policies, by using better information prediction, and 1.2%–8% less overall execution time than did the One Step and Predicted Mean Interval Scheduling policies. We also see that taking variation information into account in the scheduling policy results in more *predictable* application behavior. The History Conservative Scheduling policy exhibited 2%–32% less standard deviation of execution time than did the History Mean. The Conservative Scheduling policy exhibited 1.5%–77% less standard deviation in execution time than did the One-Step Scheduling policy and 7%–41% less standard deviation of execution time than did the Predicted Mean Interval Scheduling policy.

The second metric we used, *Compare*, is a relative metric that evaluates how often each run achieves a minimal execution time. We consider a scheduling policy to be "better" than others if it exhibits a lower execution time than another policy on a given run. Five possibilities exist: *best* (best execution time among the five policies), *good* (better than three policies but worse than one), *average* (better than two policies and worse than two), *poor* (better than one policy but worse than three), and *worst* (worst execution time of all five policies). The results indicate that

Conservative Scheduling using predicted mean and variation information is more likely to have a "best" or "good" execution time than are the other approaches on both clusters. Clearly, taking account of the average and variation CPU information during the period of application can significantly improve the application's performance.

The third metric involves using a T-test to show the significance of the improvement of our strategy over other strategies. A T-test is a statistical method used to assess whether the means of two groups are significantly different from each other [2]. The result of a T-test is a set of P-values that indicate the possibility that the differences could have happened by chance: a lower P-value means a more significant difference between two groups, so for our experiments smaller numbers are better. T-tests can be paired or unpaired – a paired T-test is used when the two groups are not independent, and an unpaired test is used when the two groups are independent. For our experiments, we calculated both paired and unpaired T-tests because it was not always clear whether the groups should be considered independent of one another. In addition, T-tests can be one-tailed, an option that is used when one group is expected to always be less than (or greater than) the other and we know that direction, or two-tailed, an option that is used only to show a difference that can sometimes be less and sometimes be greater. Since our strategy should always be better than the other strategies, we used a one-tail test. The results of the T-tests show that most P-values, especially those for paired T-tests, are below 10%. These results indicate that the possibility of the improvement happening by chance is quite small.

To summarize: Independent of the loads, CPU capabilities, application execution time, and number of resources, the Conservative Scheduling policy based on our tendency-based prediction strategy with mixed variation achieved better results than the other policies. It was

both the best policy in more situations under all load conditions on all clusters and the policy that resulted in the shortest execution time and the smallest variation in execution time.

7.2 Parallel Data Transfer Experiments

7.2.1 Experimental Methodology

We compare five scheduling policies:

(1) Best One Scheduling policy (**BOS)**: Retrieve data from the network link with the highest predicted mean bandwidth.

(2) Equal Allocation Scheduling policy (**EAS**): Retrieve the same amount of data from each source.

(3) Mean Scheduling policy **(MS)**: Allocate data according to the time balancing formula and use the interval bandwidth prediction for the effective bandwidth. This is equivalent to a tuning factor equal to 0.

(4) Nontuned Stochastic Scheduling policy (**NTSS**): Allocate data according to the time-balancing formula and use nontuned bandwidth variability to adjust the value of effective bandwidth. This is equivalent to a tuning factor equal to 1.

(5) Tuned Conservative Scheduling policy (TCS): Allocate data according to the time balancing formula, and use the tuning factor as described in Section 6.2.2 to decide how conservative the scheduling policy should be. For links with higher variability, we estimate more conservative effective bandwidth and thus allocate less data. The value of the tuning factor adapts from 0 to 1 according to the variation in bandwidth, using the formula given in Section 6.2.2.

We implemented multiple-link parallel data transfers using the partial data transfer function provided by GridFTP, part of the Globus Toolkit [19]. We measured the parallel data transfer time achieved for the five scheduling policies on different sets of machines; every set included

three source machines and one destination machine. Each machine had a replica of the file and provided part of the data, with the amount transferred from each source determined by the scheduling policy. Each pair of source and destination links opened one TCP socket. The networks may encounter contending load from other users during our experiments.

We alternated scheduling policies for the same data transfers so that any two adjacent runs experienced similar load and variation in the environment. For each set of experiments we performed approximately 100 runs, but the experimental data was consistent with larger runs on similarly loaded platforms. For methods 1 and 3–5, the effective bandwidth and the data allocation strategy were recalculated before every run using the real-time information. Complete experimental results and discussion can be found at [38], summarized in the following section.

7.2.2 Experimental Results

To compare the various policies, we again used three metrics: an absolute comparison of transfer times, a relative measure of achievements, and a statistical analysis of the significance of the improvement of our strategy. The first metric involves an average mean and an average standard deviation for all transfer times of each scheduling policy as a whole. This metric gives a rough evaluation of the performance of each scheduling policy over a given interval of time. The results show that over the entire run, the Tuned Conservative Scheduling policy exhibited 3%–51% less overall transfer time than the Best One Scheduling and Equal Allocation Scheduling policies (presumably because it takes load balancing into account) and 2% to 7% less overall transfer time than Mean and Nontuned Stochastic Scheduling policy (presumably because it takes network performance variability into account). Moreover, considering load balancing and variation information in the scheduling policy results in more predictable behavior: the Tuned Conservative Scheduling policy exhibited a 1% to 84% smaller standard deviation in transfer time than the others.

The second metric we used was *Compare*. As we had for the data-parallel experiments, we evaluated five possibilities: best, good, average, poor, and worst. The results show that Tuned Conservative Scheduling using predicted mean and tuned variation is more likely to have a "best" or "good" transfer time than are the other approaches. This fact suggests that one can significantly improve the transfer time by appropriately taking account of the average and variation network information during the period of data transfer in the scheduling policy.

The Equal Allocation Scheduling policy was always "worst" relative to the other approaches in the all but one experiments. The reason is that in these experiments, network capabilities are highly heterogeneous. Thus, the EAS strategy of allocating an equal amount of data to all sources results in "unbalanced" workload allocation and poor performance. In contrast, the Best One Scheduling policy performed worst in one experiment. The reason rests with the fact that the network capabilities are similar in this experiment, and thus load-balancing strategies that distribute load over multiple links tends to perform better than the Best One Scheduling strategy of selecting a single "best" link.

The third metric used was the T-test. For our experiments, we calculated both paired and unpaired one-tailed T-tests comparing the Tuned Conservative Scheduling strategy with the other four strategies. The results indicate that the possibility of the improvement happening by chance is small. Thus, we conclude that our Tuned Conservative Scheduling policy achieves significant improvements relative to the other three strategies in most cases.

To summarize our results: For all loads and capabilities considered, the Tuned Conservative Scheduling policy achieved better results than did the other policies considered. It was both the best policy in more situations under all load conditions and the policy that resulted in the shortest transfer time and the smallest variation in transfer time.

8 Conclusion

We have presented a conservative scheduling policy that achieves efficient execution of data-parallel applications and parallel data transfers in heterogeneous and dynamic environments. This policy uses information about the expected mean *and variance* of future resource capabilities to define data mappings appropriate for dynamic resources. Intuitively, the use of variance information is appealing because it provides a measure of resource "reliability." Our results suggest that this intuition is valid.

Our work comprises three distinct components. First, we evaluate two families of novel one-step-ahead prediction strategies. Our results show that a dynamic tendency prediction model with different ascending and descending behavior performs best among all strategies studied. A comparative study conducted on a set of 38 machine load traces shows that this new predictor achieves much better results than do other techniques. However, we found experimentally that this predictor cannot outperform NWS predictors when predicting the network capability as it did on CPU load information prediction. One possible explanation is the different autocorrelation behaviors between two adjacent measurements for CPU load and network load time series. Previous studies [11,35] reveal that the CPU load is strongly correlated over time, and the autocorrelation between two adjacent measurement could be as high as 0.95. But for most of the network capability time series, the autocorrelation function value between two adjacent observations is rather small (only between 0.8 and 0.1). Our new homeostatic and tendency-based prediction strategies, which give more weight to recent data, cannot track the trend in network capability time series well. However, NWS predictors, which take account of more statistic information, will give better prediction for these time series.

We also show how to obtain predictions of expected mean and variance information by extending the one-step-ahead time series predictors. We use our best tendency-based predictor

for CPU load information prediction and NWS predictors for future network capability prediction. But any one-step-ahead predictor that can outperform our predictors is also applicable and has the potential to further improve the efficiency of our conservative scheduling strategy.

Moreover, we show how information about expected future mean and variance (as obtained, for example, from our predictions) can be used to guide data mapping decisions. In brief, we assign less work to less reliable (higher variance) resources, thus protecting ourselves against the larger contending load spikes that we can expect on those systems. We use a conservative estimated CPU load prediction and effective bandwidth capability prediction for Cactus application and GridFTP implementation, respectively, to make data allocation decision by the time-balancing formula. Our estimation is only one possible approach. There are many ways to get the conservative resource capability estimation as long as (1) the estimated resource capability is inversely proportional to the variance of the resource capability (that is, for resource with higher variation in its performance, we have a smaller estimated effective resource capability, thus less work load); and (2) the result is reasonable (e.g., the estimated resource capability should not be an infinite large number).

We apply our prediction techniques and scheduling policy to a substantial astrophysics application and a data transfer application. Our results demonstrate that our technique can obtain better execution times and more predictable application behavior than do previous methods that focus on predicted means alone or that use variances in a less effective manner. While the performance improvements obtained are modest, they are obtained consistently and with no modifications to the application beyond those required to support nonuniform data distributions.

Acknowledgments

We are grateful to Peter Dinda for the use of his set of time series and the use of his load trace play tool. We also are grateful to our colleagues within the GrADS project for providing access to testbed resources. This work was supported in part by the Grid Application Development Software (GrADS) project of the NSF Next Generation Software program, under Grant No. 9975020, and in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract W-31-109-Eng-38.

References

[1] Math World website http://mathworld.wolfram.com/BonferroniCorrection.html.
[2] The t-Test: http://trochim.human.cornell.edu/kb/stat_t.htm.
[3] Allen, G., Benger, W., Dramlitsch, T., Goodale, T., Hege, H.-C., Lanfermann, G., Merzky, A., Radke, T., Seidel, E. and Shalf, J., Cactus Tools for Grid Applications, *Cluster Computing*, 4 (2001) 179-188.
[4] Allen, G., Benger, W., Goodale, T., Hege, H.-C., Lanfermann, G., Merzky, A., Radke, T., Seidel, E. and Shalf, J., The Cactus Code: A Problem Solving Environment for the Grid. *Proceedings of the Ninth IEEE International Symposium on High Performance Distributed Computing (HPDC9)*, Pittsburgh, 2000.
[5] Arabe, J.N.C., Beguelin, A., Loweamp, B., Seligman, E., Starkey, M. and Stephan, P., Dome: Parallel Programming in a Heterogeneous Multi-user Environment. Carnegie Mellon University, School of Computer Science, 1995.
[6] Berman, F., Chien, A., Cooper, K., Dongarra, J., Foster, I., Gannon, D., Johnsson, L., Kennedy, K., Kesselman, C., Mellor-Curmmey, J., Reed, D., Torczon, L. and Wolski, R., The GrADs Project: Software Support for High-level Grid Application Development, *The International Journal of High Performance Computing Applications*, 15 (2001) 327-344.
[7] Berman, F., Wolski, R., Figueira, S., Schopf, J. and Shao, G., Application-Level Scheduling on Distributed Heterogeneous Networks. *Supercomputing'96*, 1996.
[8] Crovella, M.E. and Bestavros, A., Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes, *IEEE/ACM Transactions on Networking*, 5 (1997) 835-846.
[9] Czajkowski, K., Foster, I., Kesselman, C., Sander, V. and Tuecke, S., SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource management in Distributed Systems. *8th Workshop On Job Scheduling Strategies for Parallel Processing*, Edinburgh, Scotland, 2002.
[10] Dail, H.J., A Modular Framework for Adaptive Scheduling in Grid Application Development Environments. *Computer Science*, University of California, California, San Diego, 2001.
[11] Dinda, P.A., The Statistical Properties of Host Load, *Scientific Programming*, 7 (1999).
[12] Dinda, P.A., Online Prediction of the Running Time of Tasks, *Cluster Computing*, 5 (2002).
[13] Dinda, P.A., A Prediction-based Real-time Scheduling Advisor. *Proceedings of the 16th International Parallel and Distributed Processing Symposium (IPDPS 2002)*, 2002.
[14] Dinda, P.A. and O'Hallaron, D.R., The Statistical Properties of Host Load. *Fourth Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers (LCR 98)*, Pittsburgh, PA, 1998, pp. 319-334.
[15] Dinda, P.A. and O'Hallaron, D.R., The Statistical Properties of Host Load. *The Fourth Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers (LCR 98)*, Pittsburgh, PA, 1998, pp. 319-334.
[16] Dinda, P.A. and O'Hallaron, D.R., Host Load Predicction Using Linear Models, *Cluster Computing*, 3 (2000).

[17] Dinda, P.A. and O'Hallaron, D.R., Realistic CPU Workloads Through Host Load Trace Playback. *Proc. 5th Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers (LCR 2000)*, Springer LNCS 1915, Rochester, NY, 2000, pp. 265-280.

[18] Figueira, S.M. and Berman, F., Mapping Parallel Applications to Distributed Heterogeneous Systems. University of Californian, San Diego, 1996.

[19] Foster, I. and Kesselman, C., The Globus Project: A Status Report. *Proc. IPPS/SPDP '98 Heterogeneous Computing Workshop*, 1998.

[20] Gehring, J. and Reinefeld, A., Mars: A Framework for Minimizing the Job Execution Time in a Metacomputing Environment, *Future Generation Computer Systems*, 12(1) (1996) 87-99.

[21] Harchol-Balter, M. and Downey, A., Exploiting Process Lifetime Distributions for Dynamic Load Balancing. *Proceedings of ACM Sigmetrics' 96 Conference on Measurement and Modeling of Computer Systems*, Philadelphia, PA, 1996, pp. 13-24.

[22] Kumar, S., Das, S.K. and Biswas, R., Graph Partitioning for Parallel Applications in Heterogeneous Grid Environments. *submitted to International Parallel and Distributed Processing Symposium (IPDPS 2002)*, Florida, 2002.

[23] Kumar, S., Maulik, U., Bandyopadhyay, S. and Das, S.K., Efficient Task Mapping on Distributed Heterogeneous System for Mesh Applications. *International workshop on Distributed Computing (IWDC 2001)*, Calcutta, India, 2001.

[24] Liu, C., Yang, L., Foster, I. and Angulo, D., Design and Evaluation of a Resource Selection Framework for Grid Applications. *Proceedings of the 11th IEEE International Symposium on High-Performance Distributed Computing (HPDC 11)*, Edinburgh, Scotland, 2002.

[25] Ripeanu, M., Iamnitchi, A. and Foster, I., Performance Predictions for a Numerical Relativity Package in Grid Environments, *International Journal of High Performance Computing Applications*, 15 (2001).

[26] Schopf, J.M., Performance Prediction and Scheduling for Parallel Applications on Multi-User Cluster. *Department of Computer Science and Engineering*, University of California San Diego, San Diego, 1998, pp. 247.

[27] Schopf, J.M., A Practical Methodology for Defining Histograms for Predictions and Scheduling. *ParCo'99*, 1999.

[28] Schopf, J.M. and Berman, F., Stochastic Scheduling. *SuperComputing'99*, Portland, Oregon, 1999.

[29] Smith, W., Foster, I. and Taylor, V., Predicting Application Run Times Using Historical Information. *Proceedings of the IPPS/SPDP'98 Workshop on Job Scheduling Strategies for Parallel Processing*, 1998.

[30] Turgeon, A., Snell, Q. and Clement, M., Application Placement Using Performance Surface. *HPDC2000*, Pittsburgh, Pennsylvania, 2000.

[31] Vazhkudai, S., Schopf, J.M. and Foster, I., Predicting the Performance of Wide Area Data Transfer. *16th Int'l Parallel and Distributed Processing Symposium (IPDPS 2002)*, Fort Lauderdale, Florida, 2002.

[32] Weissman, J.B. and Zhao, X., Scheduling Parallel Applications in Distributed Networks, *Journal of Cluster Computing*, 1 (1998) 109-118.

[33] Wolski, R., Dynamically Forecasting Network Performance Using the Network Weather Service, *Journal of Cluster Computing* (1998).

[34] Wolski, R., Spring, N. and Hayes, J., The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing, *Journal of Future Generation Computing Systems* (1998) 757-768.

[35] Wolski, R., Spring, N. and Hayes, J., Predicting the CPU availability of Time-shared Unix Systems. *Proceedings of 8th IEEE High Performance Distributed Computing Conference (HPDC8)*, Redondo Beach, CA, 1999.

[36] Yang, L., Foster, I. and Schopf, J.M., Homeostatic and Tendency-based CPU Load Predictions. *Proceedings of International Parallel and Distributed Processing Symposium (IPDPS2003)*, 2003.

[37] Yang, L., Schopf, J.M. and Foster, I., Conservative Scheduling: Using Predicted Variance to Improve Scheduling Decisions in Dynamic Environments. *Proceedings of SuperComputing 2003*, 2003.

[38] Yang, L., Schopf, J.M. and Foster, I., Improving Parallel Data Transfer Times Using Predicted Variances in Shared Networks. *Cluster Computing and Grid (CCGrid 2005)*, Cardiff, UK, 2005.

[39] Yang, Y. and Casanova, H., RUMR: Robust Scheduling for Divisible Workloads. *Proceedings of the 12th IEEE Symposium on High Performance and Distributed Computing (HPDC-12)*, Seattle, 2003.

[40] Yang, Y. and Casanova, H., UMR: A Multi-Round Algorithm for Scheduling Divisible Workloads. *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03)*, Nice, France, 2003.